Bit-Wise Arithmetic Coding for Data Compression

Aaron B. Kiely[1]

Jet Propulsion Laboratory, **MS** 238-420, 4800 Oak Grove Drive, Pasadena, **California,** 91109, USA

Abstract - **Consider the problem of compressing a uniformly quantized IID source. A traditional approach is to assign variable length codewords to the quantizer output symbols or groups of symbols (e.g., Huffman coding). Here wc propose an alternative solution: assign a *fixed length* binary codeword to each output symbol in such a way that a zero is more likely than a one iu every codeword bit position. This redundancy is then exploited using a block-adaptive binary arithmetic encoder to compress the data. This technique is simple, has low overhead, and cau be used as a progressive transmission system.**

1. Encoding Procedure

A continuous source with probability density $f(x)$ is quantized by a uniform quantizer whose output symbols are mapped to b bit codewords. The first codeword bit indicates the sign of the quantizer reconstruction point. Each successive bit gives a further level of resolution and is assigned so that zeros are more concentrated near the origin. Figure 1 illustrates this mapping for $b = 4$.

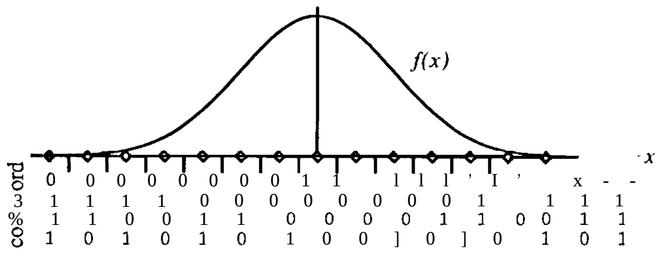

Fig. 1: Example of a pdf and codeword assignment for a four bit uniform quantizer.

We assume that $f(x)$ is symmetric about $x = 0$ and nonincreasing with $|x|$ so that the probability is more concentrated near the origin. Such sources arc not uncommon in practice. Because of this assumption, the codeword assignment ensures that a zero will be more likely than a one in every bit position.

Codewords corresponding to N adjacent source samples arc grouped together. The N sign bits of the codeword sequence arc encoded using a block-adaptive binary arithmetic encoder. Then the N next, most significant bits arc encoded, and so 011. Each bit sequence is encoded independentl - at the ith stage the arithmetic coder estimates the *unconditional* probability that the ith codeword bit is a zero. This can be viewed as a simple progressive transmission system- each subsequent codeword bit gives a further level of detail about the source.

The obvious loss is that wc lose the benefit of inter-bit dependency. E.g., the probability that the second bit is a zero is not in general independent of the value of the first bit, though the encoding procedure acts as if it were. However, for many sources (e.g., Gaussian and Laplacian), this loss is small, and this technique often has lower redundancy than Huffman coding, because the arithmetic coder is not required to produce an output symbol for every input symbol.

The independent treatment of the codeword bits provides some benefits. The overhead required increases linearly in b. By contrast, because the number of codewords is 2^b, the overhead of block-adaptive Huffman coding increases exponentially in b unless we are able to cleverly exploit additional information about the source [2].

11. Arithmetic Encoder Operation

A binary arithmetic encoder has a single parameter P, the anticipated probability of a zero. Wc encode an N-length sequence of bits block-adaptively, i.e., the encoder output sequence is preceded by overhead bits that identify to the decoder the value of 1' being used. By using $\log_2 N$ bits of overhead, we could specify the exact frequency of zeros in the sequence, but by using fewer bits we can exchange accuracy for lower overhead. If m overhead bits arm used, we can select 2'" probabilities $\{\rho_1, \rho_2, \ldots \rho_{2^m}\}$ that can be used as values for 1'. This amounts to using line segments to approximate the binary entropy function [1].

Omitting the remaining details, we find that for large N, to minimize the maximum redundancy (including overhead), the probability values arc

$$\rho_i \approx \frac{1}{2}\Big[1 - \sin\Big(\frac{\pi}{2^{m+1}}[1 + 2“’-^{22}01$$

and the optimal number of overhead bits m is approximately.

$$m \approx \frac{1}{2}\log_2 N + \log_2 \pi - 1.$$

The encoder counts the number of zeros in the input sequence to determines the probability index i. **Wc** transmit m bits to identify i, followed by the arithmetic encoder output sequence. The encoder and decoder both usc parameter $P = \rho_i$.

111. Performance

The rate R of the bit-wise arithmetic coder is approximately

$$R \approx H(Q) + \mathcal{R} + \frac{b}{N}\left[\frac{1}{2\ln 2} + \log_2 \pi - \frac{1}{2} + \frac{1}{2}\log_2 N\right]$$

here $H(Q)$ is the entropy of the quantized source and $\mathcal{R}$ is the redundancy due to independent treatment of the codeword bits.

References

[1] A. B. Kiely, "Bit-Wise Arithmetic Coding for Data Compression," *TDA Progress Reports* **42-117, pp. 145-160,** January-March 1994.

[2] R. J. McEliece and T. H. Palmatier, "Estimating the Size of Huffman Code Preambles," *TDA Progress Reports* **42-114,** April-June 1993, pp. 90-95, August 15, **1993.**

[1] The research described in this paper was performed at the Jet Propulsion Laboratory, California Institute of Technology, under contract with the National Acronautics and Space Administration.

Bit-Wise Arithmetic Coding

Aaron B. Kiely

Jet Propulsion Laboratory

Mail Stop 238-420

4800 oak Grove Drive

Pasadena, CA 91109

aaron@shannon.jpl.nasa.gov

818-354-6951

1 Introduction

Consider the problem of compressing a uniformly quantized IID source. A traditional approach is to assign variable length codewords to the quantizer output symbols or groups of symbols (e.g., Huffman Coding). Here we propose an alternative solution: assign a *fixed length* binary codeword to each output symbol in such a way that a zero is more likely than a one in every codeword bit position. This redundancy is then exploited using a block-adaptive binary arithmetic encoder to compress the data. This technique is simple, has low overhead, and can be used as a progressive transmission system.

2 Encoding Procedure

A continuous source with probability density $f(x)$ is quantized by a uniform quantizer whose output symbols are mapped to b bit codewords. The first codeword bit indicates the sign of the quantizer reconstruction point. Each successive bit gives a further level of resolution and is assigned so that zeros are more concentrated near the origin. Figure 1 illustrates this mapping for $b = 4$.

We assume that $f(x)$ is symmetric about $x = 0$ and nonincreasing with $|x|$ so that the probability is more concentrated near the origin. Such sources are not uncommon in practice. Because of this assumption, the codeword assignment ensures that a zero will be more likely than a one in every bit position.

Codewords corresponding to N adjacent source samples are grouped together. The N sign bits of the codeword sequence are encoded using a block-adaptive binary arithmetic encoder. Then the N next most significant bits are encoded, and so on. Each bit sequence is encoded independently- at the ith stage the arithmetic coder estimates the *unconditional* probability that the ith codeword bit is a zero. This can

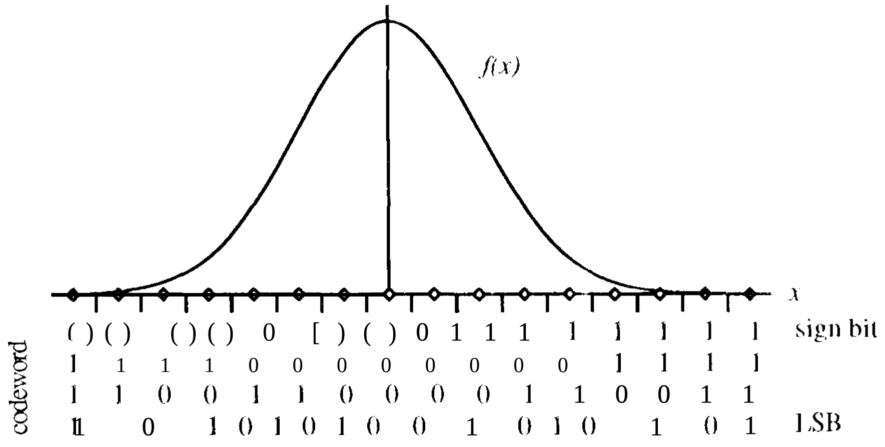

Figure 1: Example of a pdf and codeword assignment for a four bit uniform quantizer.

be viewed as a simple progressive transmission system each subsequent codeword bit gives a further level of detail about the source.

The obvious loss is that we lose the benefi - of inter-bit dependency. E.g., the probability that the second bit is a zero is not in general independent of the value of the first bit, though the encoding procedure acts as if it were. However, this loss is often small, and for many practical sources, this technique has lower redundancy than Huffman coding, because the arithmetic coder is not required to produce an output symbol for every input symbol.

The independent treatment of the codeword bits provides some benefits. As we'll see in the Section 4, the overhead required increases linearly in b. By contrast, because the number of codewords is 2^b, the overhead of block-adaptive Huffman coding increases exponentially in b unless we are able to cleverly exploit additional information about the source [2].

Another feature of this technique is that because it is progressive, it provides a simple means of handling a rate constraint: we simply encode the blocks of N bits until the allocated rate is exhausted. The distortion is automatically reduced for "more compressible" sources when the initial codeword bits can be efficiently encoded, we are able to send additional (less significant) bits, so the [illegible] resolution increases automatically.

3 Arithmetic Encoder

A binary arithmetic encoder has a single parameter P, the anticipated probability of a zero. We encode an N-length sequence of bits block-adaptively, i.e., the encoder output sequence is preceded by overhead bits that identify to the decoder the value of P being used. By using $\log_2 N$ bits of overhead, we could specify the exact frequency of zeros in the sequence, but by using fewer bits we can exchange accuracy for lower overhead. In this section we summarize results found in [1].

The average rate of a binary arithmetic encoder for an N-length sequence is

$$R_{\text{arith}} = h(P,F) + \frac{k + 1/2}{N} \tag{1}$$

where

$$h(P,F) \overset{\triangle}{=} -F\log_2 - (1-F)\log_2(1-P)$$

which is a line tangent to the binary entropy function $\mathcal{H}(F)$ at the point $F = P$, F is the fraction of bits in the sequence that are zero, and the quantity k represents the penalty from finite precision arithmetic performed by the encoder. At 9 bit resolution, simulations indicate that k is approximately equal to .32 bits.

If m overhead bits are used, we can select 2^m probabilities $\{\rho_1, \rho_2, \ldots \rho_{2^m}\}$ that can be used as values for P. From (1), we see that this amounts to using line segments to approximate the binary entropy function, as illustrated in Figure 2.

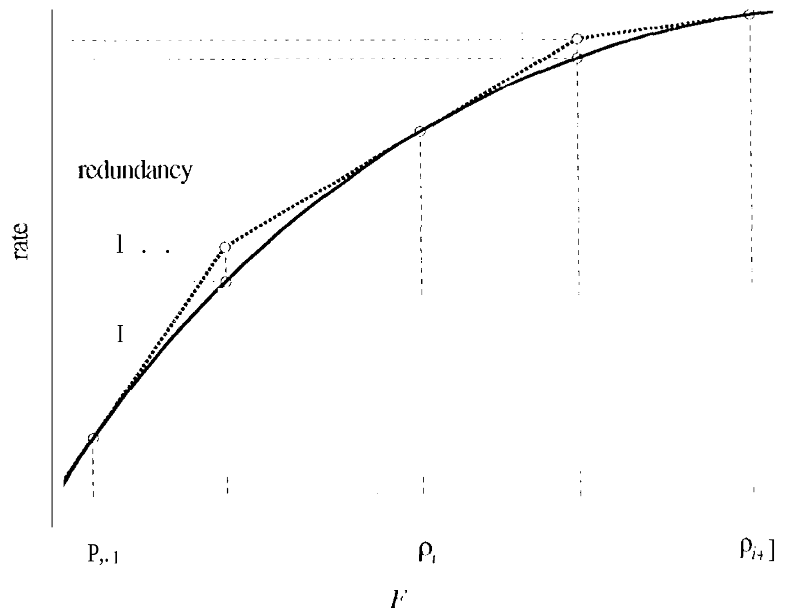

Figure 2: Binary entropy and line segment approximation.

Omitting the remaining details, we find that for large N, to minimize the maximum redundancy (including overhead), the probability values are

$$\rho_i \approx \frac{1}{2}\left[1 - \sin\left(\frac{\pi}{2^{m+1}}[1 + 2^m - 2i]\right)\right]$$

and the optimal number of overhead bits is approximately

$$m \approx \frac{1}{2}\log_2 N + \log_2 \pi - 1.$$

The encoding operation is straightforward. The encoder computes F by counting he number of zeros in the input sequence and determines the probability index i:

$$i \approx \left\lceil 2^m \left(\frac{1}{2} + \frac{\mathrm{si}}{} \frac{2F - 1)}{\pi} \right) \right\rceil$$

The encoder uses m bits to identify i, followed by the arithmetic encoder output sequence. The encoder and decoder both use parameter $P = \rho_i$.

4 Performance

The rate of the bit-wise arithmetic coder is approximately

$$R_{\mathrm{bit-arith}} \approx H(Q) + \mathcal{R} + \frac{b}{N} \left. \frac{1}{2\ln 2} + \log_2 \pi - \frac{1}{2} + k + \frac{1}{2}\log_2 N \right]$$

ere $H(Q)$ is the entropy of the quantized source and $\mathcal{R}$ is the redundancy due to independent treatment of the codeword bits

In Figure 3 we lot simulated rate-distortion curves for bit-wise arithmetic coding applied to Gaussian and Laplacian sources. For comparison, we also show the performance of block-adaptive Huffman coding and block-adaptive combined zero-runlength and Huffman coding (denoted "ZRH" in the figure).

References

[ ] A 3. Kiely, "Bit-Wise Arit metic Coding for Data Compression," *TDA Progress Reports* 42-117, pp. 145- 60, January-March 1994

[2] R J. McEliece and T Palmatier, "Estimating the Size of uffman Code Preambles," *TDA Progress Reports* 42-114, Apri -June 1993, pp. 90-95, August 15, 1993.

[3] J. J. Rissanen, "Generalized Kraft Inequality and Arithmetic Co ing," *IBM Journal of Research and Development*, pp. 198-203, May 1976.

[4] J. J. Rissanen and G. G. Langdon, Jr., "Universal Modeling and Co ing," *IEEE Trans. Inform. Theory* vol. IT-27, no. 1, pp. 12-23, January 1981.

[5] I. H. Witten, R. M. Neal, J. G. Cleary, "Arithmetic Coding for Data Compression," *Communications of the ACM* vol. 30, no. 6, pp. 520-540, June 1987.

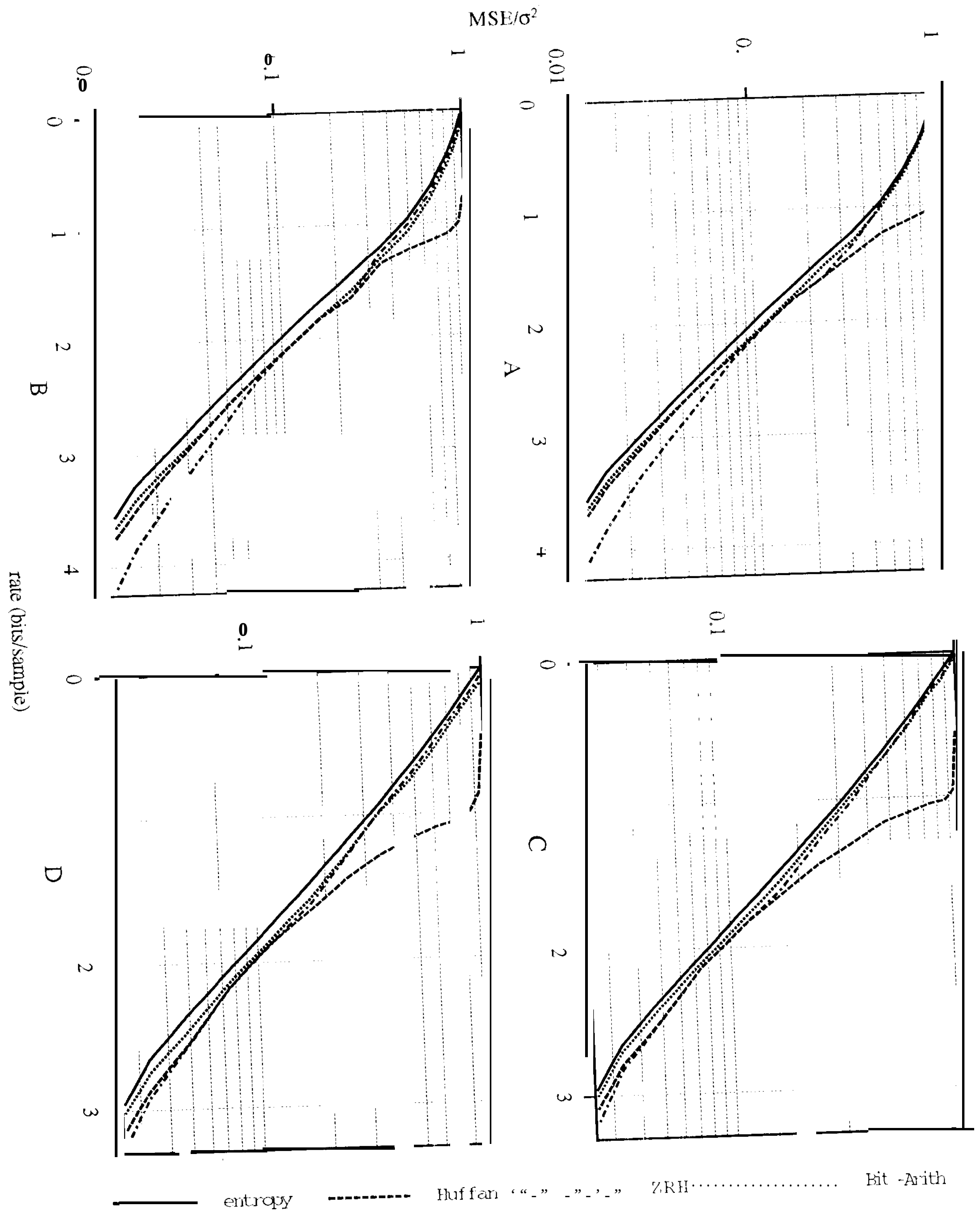

Figure 3: Performance of bit-wise arithmetic, Huffman, and combined zero-runlength and Huffman coding. a) Gaussian source, $b = 4, N = 512$, b) Gaussian source, $b = 4, N = 256$, c) Laplacian source, $b = 4, N = 512$. d) Laplacian source, $b = 4, N = 256$.